12-22-00

A

Please type a plus (+) inside this box →

PTO/SB/29 (12/97)
Approved for use through 09/30/00. OMB 0651-0032
Patent and Trademark Office: U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

UTILITY PATENT APPLICATION TRANSMITTAL

(Only for new nonprovisional applications under 37 CFR 1.53(b))

Attorney Docket No.	10734-003-999
Total Pages	73
First Named Inventor or Application Identifier	BRENDAN O'REILLY, CHRISTOPHER CHEDGEY
Express Mail Label No.	EL 501 636 358 US

APPLICATION ELEMENTS

See MPEP chapter 600 concerning utility patent application contents.

ADDRESS TO: **Assistant Commissioner for Patents, Box Patent Application, Washington, DC 20231**

1. ☒ Fee Transmittal Form
   Submit an original, and a duplicate for fee processing)
2. ☒ Specification [*Total Pages* 55]
   (preferred arrangement set forth below)
   - -Descriptive title of the Invention
   - -Cross Reference to Related Applications
   - -Statement Regarding Fed sponsored R&D
   - -Reference to Microfiche Appendix
   - -Background of the Invention
   - -Brief Summary of the Invention
   - -Brief Description of the Drawings *(if filed)*
   - -Detailed Description of the Invention (including drawings, *if filed*)
   - -Claim(s)
   - -Abstract of the Disclosure
3. ☒ Drawing(s) *(35 USC 113)* [*Total Sheets* 14]
4. ☒ Oath or Declaration (unexecuted) [*Total Sheets* 2]
   a. ☐ Newly executed (original or copy)
   b. ☐ Copy from a prior application (37 CFR 1.63(d))
   (for continuation/divisional with Box 17 completed)
   [Note Box 5 below]
   i. ☐ DELETION OF INVENTORS(S)
   Signed statement attached deleting inventor(s) named in the prior application, see 37 CFR 1.63(d)(2) and 1.33 (b).
5. ☐ Incorporation By Reference *(useable if Box 4b is checked)*
   The entire disclosure of the prior application, from which a copy of the oath or declaration is supplied under Box 4b, is considered as being part of the disclosure of the accompanying application and is hereby incorporated by reference therein.
6. ☐ Microfiche Computer Program *(Appendix)*
7. ☐ Nucleotide and/or Amino Acid Sequence Submission
   (if applicable, all necessary)
   a. ☐ Computer Readable Copy
   b. ☐ Paper Copy (identical to computer copy)
   c. ☐ Statement verifying identity of above copies

ACCOMPANYING APPLICATION PARTS

8. ☐ Assignment Papers (cover sheet & document(s))
9. ☐ 37 CFR 3.73(b) Statement ☐ Power of Attorney
   (when there is an assignee)
10. ☐ English Translation Document *(if applicable)*
11. ☐ Information Disclosure Statement (IDS)/PTO-1449 ☐ Copies of IDS Citations
12. ☐ Preliminary Amendment
13. ☒ Return Receipt Postcard (MPEP 503)
    (Should be specifically itemized)
14. ☐ Small Entity ☐ Statement filed in prior application, Statement(s) Status still proper and desired
15. ☐ Certified Copy of Priority Document(s)
    (if foreign priority is claimed)
16. ☐ Other: .

17. **If a CONTINUING APPLICATION,** *check appropriate box and supply the requisite information:*
    ☐ Continuation ☐ Divisional ☐ Continuation-in-part (CIP) of prior application No: . filed.

18. CORRESPONDENCE ADDRESS

☒ Customer Number or Bar Code Label 20583 *(Insert Customer No. or Attach bar code label here)* or ☐ Correspondence address below

NAME					
ADDRESS					
CITY		*STATE*		*ZIP CODE*	
COUNTRY		*TELEPHONE*		*FAX*	

Burden Hour Statement: This form is estimated to take 0.2 hours to complete. Time will vary depending upon the needs of the individual case. Any comments on the amount of time you are required to complete this form should be sent to the Chief Information Officer, Patent and Trademark Office, Washington, DC 20231. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Assistant Commissioner for Patents, Box Patent Application, Washington, DC 20231.

Express Mail No.:EL 501 636 358 US

PENNIE & EDMONDS LLP
COUNSELLORS AT LAW
1155 Avenue of the Americas
New York, N.Y. 10036-2711
(212) 790-9090

ATTORNEY DOCKET NO. 10734-003-999 Date: December 20, 2000

Assistant Commissioner for Patents
Box PATENT APPLICATION
Washington, D.C. 20231

Sir:

The following utility patent application is enclosed for filing:

Applicant(s): Christopher Chedgey, Brendan O'Reilly Executed on: 12/20/2000

Title of Invention: SYSTEM AND METHOD FOR COMPUTER-AIDED GRAPH-BASED DEPENDENCY ANALYSIS

PATENT APPLICATION FEE VALUE

TYPE	NO. FILED	LESS	EXTRA	EXTRA RATE	FEE
Total Claims	14	-20	0	$18.00 each	$ 0.00
Independent	5	-3	2	$80.00 each	$ 160.00
				Minimum Fee	$ 710.00
				Multiple Dependency Fee If Applicable ($270.00)	$ 270.00
				Total	$ 1,140.00
				50% Reduction for Independent Inventor, Nonprofit Organization or Small Business Concern (a verified statement as to the applicant's status is attached)	- $ 0.00
				Total Filing Fee	$ 1,140.00

☒ Priority of application no. 991070 filed on Dec. 20, 1999 in Ireland is claimed under 35 U.S.C. § 119.
☐ The certified copy of the priority application has been filed in application no. filed
☐ Amend the specification by inserting before the first line the following sentence: This is a continuation-in-part of application no. filed .

Please charge the required fee to Pennie & Edmonds LLP Deposit Account No. 16-1150. A copy of this sheet is enclosed.

Respectfully submitted,

37,242

Garland T. Stephens (Reg. No.)
PENNIE & EDMONDS LLP

Enclosure

This form is not for use with continuation, divisional, re-issue, design or plant patent applications.

Appendix 1 – Methods of class HiGraph

getRoot
public HiNode getRoot()
returns a HiNode object which forms the root of the node tree
guaranteed to exist and cannot be deleted or moved

---

getConnectionMap
public headway.util.ICollection getConnectionMap()
returns a (readonly) map containing all essential connections present in the graph

---

addConnection
public boolean addConnection(HiNode fromNode,
HiNode toNode)
Deprecated. create a new connection in the graph from the fromNode to the toNode

---

addConnection
public boolean addConnection(HiConnection hc)
create a new connection in the graph from the fromNode to the toNode

---

removeConnection
public void removeConnection(HiConnection hc,
boolean flush)
remove the passed HiConnection from the graph. Equivalent to calling remove() on the HiConnection

---

getNodeMap
public headway.util.ICollection getNodeMap()
returns a (readonly) collection containing all the nodes present in the graph irrespective of the nodetree structure
Note that this does not include the root node

---

findNode
public HiNode findNode(long id)
utility method to locate a specific node by key (equivalent to calling getElement() on the graph's NodeMap)
Returns the corresponding HiNode object or null if none exists

---

findNodeByName
public HiNode findNodeByName(java.lang.String name)
utility method which returns the first node found which matches the name passed or null if none found.
Since node names are not guaranteed to be unique, this method should be used with extreme care. It is also highly inefficient in that it needs to iterate all nodes in the graph doing a string compare on each.

---

findFreeNodeName
public java.lang.String findFreeNodeName(java.lang.String prefix)
method which finds the first unused node name with the given prefix e.g. passing "cluster" might return cluster 1, cluster 2, etc.

---

removeNode
public void removeNode(HiNode hn,
boolean flush)

---

clear
public void clear()
clears the graph i.e. removes all nodes and connections

---

flush
public void flush()
clears the graph i.e. removes all nodes and connections
Specified by:
flush in interface headway.util.Flushable

---

getDirty
public boolean getDirty()

---

setDirty
public void setDirty(boolean dirty)

---

setDriver
public void setDriver(HGDriver driver)

---

getDriver
public HGDriver getDriver()

Appendix 2 – Fields and Methods of class HiNode

Fields:

pathSeparator
public static java.lang.String pathSeparator
separation character used for path and fully qualified name

ID_NEW
public static final long ID_NEW

ID_ROOT
public static final long ID_ROOT

STATE_UNRESOLVED
public static final int STATE_UNRESOLVED
State indicating that node has not yet been resolved (parsed)

STATE_RESOLVED
public static final int STATE_RESOLVED
State indicating that node has been successfully resolved (parsed)

STATE_UNRESOLVABLE
public static final int STATE_UNRESOLVABLE
State indicating that node could not be resolved
Information on the nature of the reolve failure can be obtained by calling
getResolveFailureId

STATE_MISSING
public static final int STATE_MISSING

Indicates that node is missing (and therefore unresolvable)

STATE_WRONG_LOCATION
public static final int STATE_WRONG_LOCATION
Indicates that node has no valid driver and is therefore not resolvable

---

STATE_NO_DRIVER
public static final int STATE_NO_DRIVER
Indicates that node has no valid driver and is therefore not resolvable

---

ATTRIB_PUBLIC
public static final int ATTRIB_PUBLIC

ATTRIB_PRIVATE
public static final int ATTRIB_PRIVATE

ATTRIB_PROTECTED
public static final int ATTRIB_PROTECTED

ATTRIB_STATIC
public static final int ATTRIB_STATIC

ATTRIB_FINAL
public static final int ATTRIB_FINAL

ATTRIB_SYNCHRONIZED
public static final int ATTRIB_SYNCHRONIZED

ATTRIB_THREADSAFE
public static final int ATTRIB_THREADSAFE

ATTRIB_TRANSIENT
public static final int ATTRIB_TRANSIENT

ATTRIB_NATIVE
public static final int ATTRIB_NATIVE

ATTRIB_INTERFACE
public static final int ATTRIB_INTERFACE

ATTRIB_ABSTRACT
public static final int ATTRIB_ABSTRACT

---

comparator
public HiNodeComparator comparator

Methods:

resolve
public int resolve()
Triggers a parse of the underlying object and loading of connections

---

canHaveChildren
public boolean canHaveChildren()
Returns true if this HiNode can contain children

---

isMeta
public boolean isMeta()
Returns true if this node is a metanode, i.e. cannot hold direct connections

---

isClass
public boolean isClass()

---

setId
protected void setId(long id)
sets the id of this node

---

setGraph
protected void setGraph(HiGraph hg)
sets the graph in which this node lives

---

setType
public void setType(int type)
sets the type of the node - the value corresponds to a constant defined in NodeFactory

---

setUML
public void setUML(java.lang.String uml)
sets a UML respresentation of this node

---

getUML
public java.lang.String getUML()
returns a UML-style respresentation of this node

---

setAttributes
public void setAttributes(int attributes)

takes a bitmask value containing information on various attributes of the node. details see ACC constants

---

setParent
protected void setParent(HiNode hn)
sets the parent of this node

---

getType
public int getType()
returns the type of the node - the value corresponds to a constant defined in NodeFactory use getTypeString() for a string version

---

getTypeString
public java.lang.String getTypeString()
returns a string indicating the type of the node - the value corresponds to a constant defined in NodeFactory

---

getParent
public HiNode getParent()
returns the parent of the node (null if this is the root node)

---

setName
public void setName(java.lang.String sName)
sets the name of this node

---

getName
public java.lang.String getName()
returns the name of this node

---

setSourceFile
public void setSourceFile(java.lang.String sourceFile)
sets the source file name of this node

---

getSourceFile
public java.lang.String getSourceFile()
returns the source file name of this node

---

setState
public void setState(int state)
sets the state of this node

---

getState
public int getState()
returns the state of this node

---

getAttributes
public int getAttributes()
returns a bitmask containing information on various attributes of the node. details see ACC constants

---

getAttribute
public boolean getAttribute(int bit)
returns a boolean indicating whether the passed attribute flag (or flags) are set in the attributes mask

---

getId
public long getId()
returns the unique id of this node

---

getKey
public java.lang.Object getKey()
returns a string key unique to this node (stringified id)
Specified by:
getKey in interface headway.util.CollectionMember

---

isRoot
public boolean isRoot()
returns true if this is the root node (i.e. parent is null)

---

getPath
public java.lang.String getPath()
returns the path of this node (e.g. java.lang)

---

getFQName
public java.lang.String getFQName()
returns the fully qualified name of this node (e.g. if the name is String and the path is /java/lang then the fully qualified name is /java/lang/String)

---

getGraph
public HiGraph getGraph()
returns the graph in which this node resides

---

getUserObject
public java.lang.Object getUserObject()
returns a user-defined object saved with the setUserObject() method

---

setUserObject
public void setUserObject(java.lang.Object obj)
used to store a reference to an arbitrary object

---

equals
public boolean equals(HiNode hn)
true if the passed node is the same as this

---

isSibling
public boolean isSibling(HiNode hn)
returns true if the passed HiNode is a sibling of this node, i.e. shares the same parent

---

toString
public java.lang.String toString()
returns the fully qualified name of the node
Overrides:
toString in class java.lang.Object

---

addChild
public HiNode addChild(HiNode hn)
creates and returns a new child node of the requested type
Parameters:
int - type as defined in NodeFactory
long - known id of an existing object or ID_NEW (-1) for a new object

---

getChildren
public headway.util.ICollection getChildren()
returns a JCollection of the children of this node
throws an IllegalSomething exception if this node is a leaf

---

getChildren
public headway.util.ICollection getChildren(HiNodeComparator comparator)
returns a JCollection of the children of this node
throws an IllegalSomething exception if this node is a leaf

---

moveTo
public void moveTo(HiNode hn,
boolean flush)

---

getConnections
public headway.util.ICollection getConnections(int direction)
returns a collection of connections for this node
Parameters:
int - the requested direction HiEdge.TO or HiEdge.FROM
int - mode MODE_CONNECTION_DIRECT or MODE_CONNECTION_ANY

---

getRelationship
public Relationship getRelationship(HiNode hn,
int direction)
returns a Relationship object describing the relationship between this node and the passed node for the given direction. a null relationship is signified by the isEmpty flag in the relationship object. A null pointer is returned if the passed node is itself null or if it is equal to "this" node

---

getChildRelationships
public headway.util.ICollection getChildRelationships()
convenience wrapper function which returns all the relationships between the child nodes of this node (which must be capable of having children)

---

getConnectionCount
public int getConnectionCount(int direction)
returns the total number of (potentially rolled up) connections which apply to this node in the given direction

---

countConnections

protected int countConnections(int runningTotal,
int direction,
HiNode ancestor)
counts all connections with recursive calls if an ancestor node is supplied, connections where the referenced node is a descendant of the ancestor are ignored

---

flush
public void flush()
clears the graph i.e. removes all nodes and connections
Specified by:
flush in interface headway.util.Flushable

---

findCommonAncestor
public HiNode findCommonAncestor(HiNode hn)
Utility treewalking method which takes a HiNode argument and returns the (nearest) common ancestor. guaranteed to return a node (possibly root) provided that neither of this node nor the argument node passed is itself the root, in which case an IllegalArgumentException is thrown

Examples: this = root.java.util.Vector
hn = root.java.lang.String
ret = root.java
this = root.a.b.c.d
hn = root.a.b.e.f.g.h.i
ret = root.a.b

---

findFirstDescendant
public HiNode findFirstDescendant(HiNode hn)
Utility treewalking method which takes a HiNode argument which is assumed to be a descendant of this node and find the first descendant of this node which is also an ancestor of the passed node
returns null if the node passed is not a descendant of this node

Examples: this = root.java
hn = root.java.lang.String
ret = root.java.lang

this = root.a.b
hn = root.a.b.e.f.g.h.i
ret = root.a.b.e

---

isDescendantOf
public boolean isDescendantOf(HiNode hn)

---

setFQEntityName
public void setFQEntityName(java.lang.String fqentityname)

---

getFQEntityName
public java.lang.String getFQEntityName()

---

getEntityName
public java.lang.String getEntityName()

---

getEntityPath
public java.lang.String getEntityPath()

Appendix 3 – Methods of the MetaNode class

resolve
public int resolve()
Description copied from class: HiNode
Triggers a parse of the underlying object and loading of connections
Overrides:
resolve in class HiNode

---

resolve
public int resolve(boolean recursive)

---

canHaveChildren
public boolean canHaveChildren()
Description copied from class: HiNode
Returns true if this is can contain children
Overrides:
canHaveChildren in class HiNode

---

isMeta
public boolean isMeta()
Description copied from class: HiNode
Returns true if this node is a metanode, i.e. cannot hold direct connections
Overrides:
isMeta in class HiNode

---

getState
public int getState()
Description copied from class: HiNode
returns the state of this node
Overrides:
getState in class HiNode

Appendix 4– Methods of the ClassNode class

canHaveChildren
public boolean canHaveChildren()
Description copied from class: HiNode
Returns true if this is can contain children
Overrides:
canHaveChildren in class HiNode

---

isMeta
public boolean isMeta()
Description copied from class: HiNode
Returns true if this node is a metanode, i.e. cannot hold direct connections
Overrides:
isMeta in class HiNode

---

isClass
public boolean isClass()
Overrides:
isClass in class HiNode

---

isApplicationClass
public boolean isApplicationClass()

---

getFQSourceFile
public java.lang.String getFQSourceFile()

---

resolve
public final int resolve()
Description copied from class: HiNode
Triggers a parse of the underlying object and loading of connections
Overrides:
resolve in class HiNode

Appendix 5 – Methods of the FieldNode Class

canHaveChildren
public boolean canHaveChildren()
Description copied from class: HiNode
Returns true if this is can contain children
Overrides:
canHaveChildren in class HiNode

---

isMeta
public boolean isMeta()
Description copied from class: HiNode
Returns true if this node is a metanode, i.e. cannot hold direct connections
Overrides:
isMeta in class HiNode

---

getState
public int getState()
Description copied from class: HiNode
returns the state of this node
Overrides:
getState in class HiNode

Appendix 6 – Methods of the MethodNode Class

canHaveChildren
public boolean canHaveChildren()
Description copied from class: HiNode
Returns true if this is can contain children
Overrides:
canHaveChildren in class HiNode

---

isMeta
public boolean isMeta()
Description copied from class: HiNode
Returns true if this node is a metanode, i.e. cannot hold direct connections
Overrides:
isMeta in class HiNode

---

getState
public int getState()
Description copied from class: HiNode
returns the state of this node
Overrides:
getState in class HiNode

Appendix 7 – Data Fields, Constructors, and Methods of HiEdge class

Fields:

DEPENDENCY
public static final int DEPENDENCY

AGGREGATION
public static final int AGGREGATION

IMPLEMENTS
public static final int IMPLEMENTS

EXTENDS
public static final int EXTENDS

FROM
public static final int FROM

TO
public static final int TO

Constructors:

HiEdge
public HiEdge(HiNode fromNode,
HiNode toNode)

HiEdge
public HiEdge(HiNode hn1,
HiNode hn2,

int direction)

Methods:

makeKey
public static java.lang.Long makeKey(HiNode fromNode,
HiNode toNode)

---

makeKey
public static java.lang.Long makeKey(HiNode hn1,
HiNode hn2,
int direction)

---

getNode
public HiNode getNode(int direction)
returns the referenced node for the given direction

---

getToggledNode
public HiNode getToggledNode(int direction)
returns the referenced node for the given direction

---

getKey
public java.lang.Object getKey()
returns a unique key for this object
implementation is a Long DWord-style long containing the fromNode id in the low word and the toNode id in the high word

Specified by:
getKey in interface headway.util.CollectionMember

---

getGraph
public HiGraph getGraph()
returns the graph in which the edge resides

---

toString
public java.lang.String toString()
Overrides:
toString in class java.lang.Object

---

getType
public int getType()

---

setType
public void setType(int type)

---

isDependency
public boolean isDependency()

---

isAggregation
public boolean isAggregation()

---

isImplements
public boolean isImplements()

---

isExtends
public boolean isExtends()

---

getVerboseName
public java.lang.String getVerboseName()
returns a longwinded string version of the edge in the form A-->B, used mainly for debug purposes

---

toggleDirection
public static int toggleDirection(int direction)

---

getNodeId
public long getNodeId(int direction)

Appendix 8 - Methods of Relationship Class

Methods:

isMeta
public boolean isMeta()

---

getCarriedConnections
public int getCarriedConnections()

---

isEmpty
public boolean isEmpty()